BACHELORARBEIT

Implementation and Evaluation of a Support Vector Machine on an 8-bit Microcontroller

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Bachelor of Science

unter der Leitung von

Univ.Ass. Dipl.-Ing. Dr.techn. Wilfried Elmenreich
Institut für Technische Informatik
Fakultät für Informatik
Technische Universität Wien

durchgeführt von

Thomas Nowak
Matr.-Nr. 0425201
Sturzgasse 1C/10, A-1140 Wien

Wien, im Juli 2008 .....

Implementation and Evaluation of a Support Vector Machine on an 8-bit Microcontroller

Support Vector Machines (SVMs) can be used on small microcontroller units (MCUs) for classifying sensor data. We investigate the theoretical foundations of SVMs, we prove in particular a very general version of Mercer's theorem, and review the software package μSVM, which is an implementation of an SVM for use on MCUs. Present SVM solutions are not applicable to MCUs, because they need too much memory space which can be expected when running on a personal computer, but not on an MCU. It is shown that, while μSVM's execution time does not scale very well to a large number of training examples, it is possible to prematurely terminate the training process and still retain good numerical accuracy.

Contents

1. Introduction

Support Vector Machines (SVMs) are a method in machine learning where the goal is a binary classification of input data. The procedure is the following: In a first phase, the so-called *training phase*, a set of labeled objects is presented to the machine. The labels are taken from a two-element set (e. g., 0/1, A/B, $+1/-1$, good/bad, ...). The machine's objective is to derive a decision procedure from this *training set* such that it can classify correctly objects presented to it in the future. Of course, it is the supervisor's responsibility to choose the training set adequately, i. e., taking as representative objects as possible to enable good classification performance on future observations.

Support Vector Machines are used for a variety of classification tasks. These include handwriting recognition, speaker identification, text categorization and face detection.

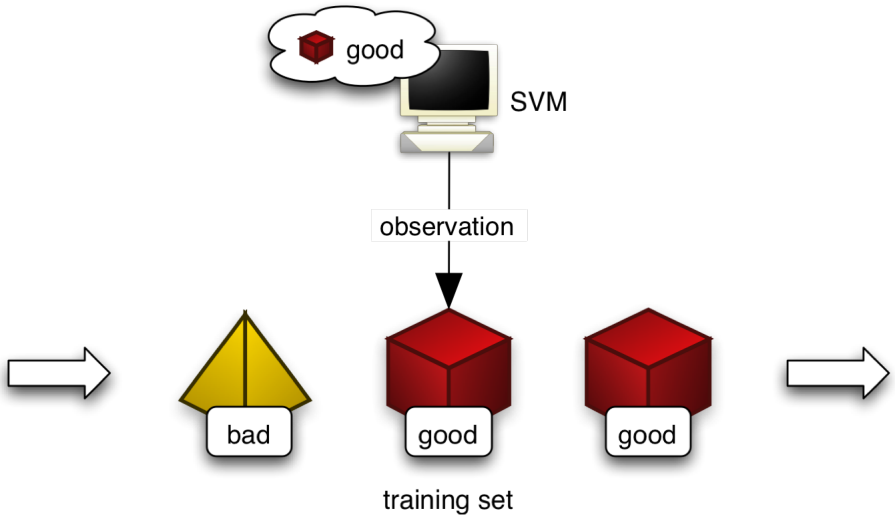

Figure 1.1.: SVM Training

1.1. Motivation

In a technical device, e. g., a control loop, it is necessary to have some sort of input data from the environment. These values are the output of *sensor devices*. Now, relying on a single sensor is not reasonable since the sensor may fail or behave highly differently in different environmental situations (e. g., temperature, light, movement). To compensate for this, sensors are replicated (not necessarily identically) and even completely different types of sensors may be used. Consider, for example, the velocity sensor of a roller coaster wagon. It might be known that velocity sensor A works well in the temperature range of 0°C to 20°C and that velocity sensor B is more accurate from 20°C onwards. So, it would be reasonable to install both sensors A and B and additionally put a temperature sensor on the device. In that way, one could use sensor A up to 20°C and sensor B above. The value returned by the sensor device will be that of sensor A or sensor B depending on the temperature. It is transparent to the control application which sensors are used and even which sensors exist in the device. The calculation[1] of the returned value is done by a microcontroller unit (MCU) inside the sensor device.

Now, it could be possible that the choice which sensor to trust is based on more than one parameter. Then the decision which sensor to use would no longer be as simple as the check whether one number is smaller than the other, but a more sophisticated method would have to be used. This problem can be tackled with the use of an SVM.

Of course, the previous example alone does not justify an implementation of the SVM *training* algorithm on the MCU, because most often, the characteristics of the sensors are known beforehand and the training can happen offline on a PC. But situations may occur where such a priori knowledge is not accessible, for example if the device is to be used in a broad variety of different environments which are not determined a priori and the device has to adjust to a new environment periodically.

Other applications for an SVM implementation on an MCU might include classification tasks like determining a "safe state" in an execution[2] or image classification (though a pixel-by-pixel comparison will not be feasible).

[1] It should be noted that this calculation need not be as easy as choosing the right sensor. It might involve taking different sensor values with appropriate weighting factors as well as more complex methods. The use of such methods is referred to as *sensor fusion*.

[2] A *safe state* could be one where it is possible for the device to enter a sleep mode, because it will not be used for a while. Of course, this decision will be a probabilistic one and hence is not suitable for safety critical applications.

1.2. Structure of the Thesis

Chapter 2 gives a textbook-like introduction to the basic concepts of the branch of optimization theory needed for SVMs. Chapter 3 derives basic algorithmic concepts in SV training and reviews popular approaches. The implementation of μSVM is described in chapter 4. Chapter 5 states results concerning the performance of μSVM. The thesis ends with a conclusion in chapter 6.

2. Optimization and SVMs

This chapter treats the mathematical foundations of Support Vector Machines. We start with the classification of the main problem in Support Vector training – namely solving the quadratic program that yields the separating hypersurface for our training set. We then investigate optimality conditions for the optimization problem and give first ideas for efficient algorithms. We end the chapter with a slight generalization of the problem which is of greatest importance in practice (kernel functions instead of scalar product).
The chapter is essentially self-contained although many proofs are left out in order to avoid an overly mathematical bias. Only some of the most prototypical (or short) proofs are stated.

2.1. Basic Optimization Theory

Definition 1. *Let $f : X \to \mathbb{R}$ be a function (the* objective function*) and $S \subset X$ be a nonempty set (the* feasibility set*). The problem of finding an $x_0 \in S$ such that*

$$f(x_0) \leqslant f(x) \quad \text{for all } x \in S \tag{2.1}$$

is called a minimization problem. □

We note that trying to solve this problem only makes sense if $\inf\{f(x) \mid x \in S\}$ exists, i. e., f is bounded from below on S.
Analogously, if we exchange "$\leqslant$" by "$\geqslant$" in Definition 1, we get a *maximization problem*. We call both an *optimization problem*. We will only be concerned with minimization problems here as the other case is analogous – we just have to flip a few inequality signs, exchange min by max, inf by sup, etc.
The problem that we will try to solve in SVM training is a so-called *convex* optimization problem, i. e., both the objective function f and the feasibility set S are convex. We will define these terms and derive some first properties.

For the following, we set

$$\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty\} \tag{2.2}$$

with

$$x < +\infty \text{ and } x + \infty = +\infty \tag{2.3}$$

for every real x. Further, X will always denote a subset of a finite-dimensional Hilbert space H over the reals. We can think $X \subset \mathbb{R}^n$ here.

2.1.1. Convexity – Definition and Simple Properties

Definition 2. *Let $A \subset X$. We call A* convex *if for all $x, y \in A$ and $0 < \lambda < 1$:*

$$\lambda x + (1 - \lambda)y \in A \tag{2.4}$$

□

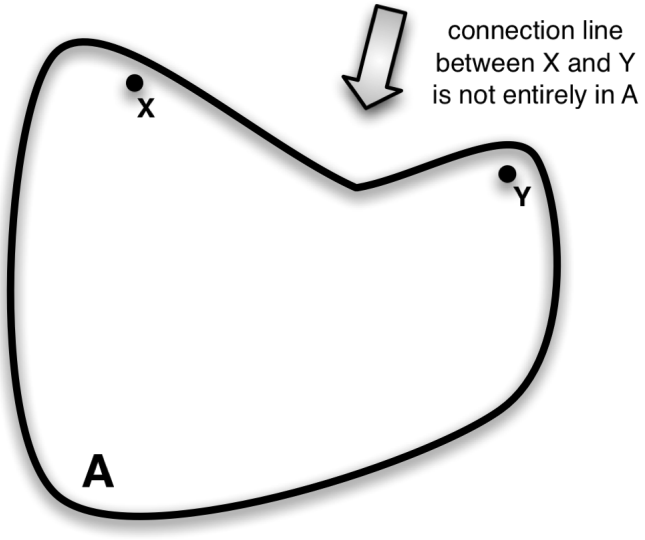

Figure 2.1.: A non-convex set

Definition 3. *Let X be convex, $f : X \to \overline{\mathbb{R}}$. We call f* convex *if for all $x, y \in X$ and $0 < \lambda < 1$:*

$$f(\lambda x + (1 - \lambda)y) \leqslant \lambda f(x) + (1 - \lambda) f(y) \tag{2.5}$$

We call f closed *if it is lower semi-continuous, i. e., for all $c \in \mathbb{R}$, $f^{-1}((c, \infty])$ is open in X.* □

Convex sets and functions turn out to be quite convenient:

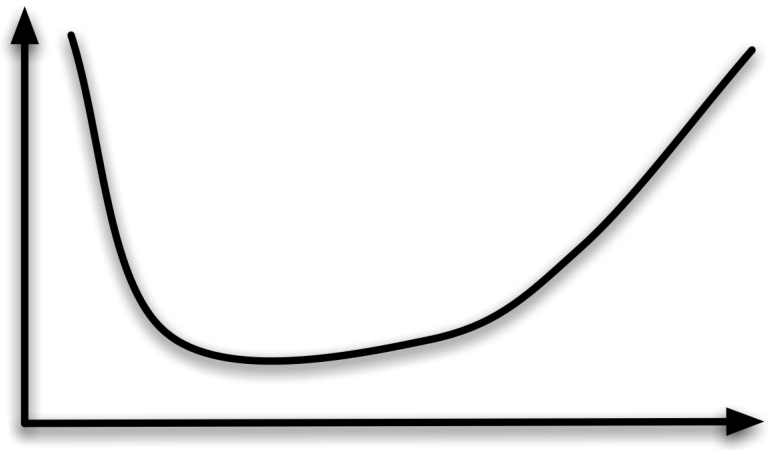

Figure 2.2.: A convex function

Proposition 1. *Let I be an arbitrary set and let S_i be a convex set for $i \in I$. Then the intersection*

$$S = \bigcap_{i \in I} S_i \tag{2.6}$$

is again convex.

Proof. For all $x, y \in S$ and all $i \in I$ holds $x, y \in S_i$. □

Proposition 2. *Let I be an arbitrary set. Let $f_i : X \to \overline{\mathbb{R}}$ be a convex function and $\lambda_i \geqslant 0$ for every $i \in I$ with at most finitely many $\lambda_i \neq 0$. Then holds:*

(1) *The function $f : X \to \overline{\mathbb{R}}$,*

$$f(x) = \sum_{i \in I} \lambda_i f_i(x) \tag{2.7}$$

is convex. If all f_i are closed, then so is f.

(2) *The function $g : X \to \overline{\mathbb{R}}$,*

$$g(x) = \sup_{i \in I} f_i(x) \tag{2.8}$$

is convex. If all f_i are closed, then so is g.

(3) *If $(I, \preceq)$ is a directed set and the pointwise limit (proper or improper towards $+\infty$) of $(f_i)_{i \in I}$ exists, then the function $h : X \to \overline{\mathbb{R}}$,*

$$h(x) = \lim f_i(x) \tag{2.9}$$

is convex.

Proof. (1): It is obvious that f is convex. We show the closedness in two steps ($\lambda > 0$, f closed $\Rightarrow \lambda f$ closed; f, g closed $\Rightarrow f + g$ closed).
It is $(\lambda f)^{-1}((c, \infty]) = f^{-1}((c/\lambda, \infty])$. Let $x \in (f+g)^{-1}((c, \infty])$, i. e.,

$$f(x) + g(x) > c. \tag{2.10}$$

We can find an $\varepsilon > 0$ such that

$$f(x) - \varepsilon + g(x) - \varepsilon > c \tag{2.11}$$

still holds. It is

$$x \in f^{-1}((f(x) - \varepsilon, \infty]) \cap g^{-1}((g(x) - \varepsilon, \infty]) \subset (f+g)^{-1}((c, \infty]) \tag{2.12}$$

with the inner set being open as the intersection of two open sets.

(2): An easy reformulation of the definition of convexity (2.5) is the following: The set

$$E(f) = \{(x, y) \in X \times \mathbb{R} \mid f(x) \leqslant y\} \tag{2.13}$$

is convex in $X \times \mathbb{R}$. ($E(f)$ is the so-called *epigraph* of f.) Now, all $E(f_i)$ are convex and thus,

$$E(g) = \bigcap_{i \in I} E(f_i) \tag{2.14}$$

is also convex by Proposition 1, i. e., g is convex. It is g closed, because

$$g^{-1}((c, \infty]) = \bigcup_{i \in I} f_i^{-1}((c, \infty]) \tag{2.15}$$

is open.

(3): Let $x, y \in X$, $\lambda \in (0, 1)$. It is

$$f_i(\lambda x + (1-\lambda)y) \leqslant \lambda f_i(x) + (1-\lambda) f_i(y) \tag{2.16}$$

for all $i \in I$ and thus, by taking the limit (non-strict inequalities are preserved under limit-taking and the field operations in $\mathbb{R}$ are continuous),

$$h(\lambda x + (1-\lambda)y) \leqslant \lambda h(x) + (1-\lambda) h(y). \tag{2.17}$$

□

We remark that the limit function h is *not* closed in general if the f_i are. This is because even limits of sequences of *continuous* functions can behave nastier than being "just semi-continuous".
(1) and (3) together imply that also the limit of the series

$$\sum_{i \in I} \lambda_i f_i \tag{2.18}$$

is convex if the sum exists. We can drop the requirement that only finitely many λ_i are nonzero here.

2.1.2. Global/Local Minima

Our search for solutions of the minimization problem will become a lot easier with the following theorem. Namely, we can restrict ourselves to search for *local* minima, because all local minima of convex functions are already global.

Theorem 1. *Let $f : X \to \overline{\mathbb{R}}$ be a convex function and $x_0 \in X$ with $f(x_0) \in \mathbb{R}$ which is a* local minimum *of f, i. e., there exists a neighbourhood U of x_0 such that*

$$f(x_0) \leqslant f(x) \text{ for all } x \in U. \tag{2.19}$$

Then x_0 is a global minimum of f.

Proof. Let x_1 be a point in X with $f(x_0) > f(x_1)$ and let $\varepsilon > 0$ such that $B_\varepsilon(x_0) \subset U$. It is $x_1 \notin B_\varepsilon(x_0)$. We set

$$\lambda = \frac{\varepsilon}{2\,\|x_1 - x_0\|}. \tag{2.20}$$

Then we have that $\lambda x_1 + (1-\lambda)x_0 \in U$ and thus

$$f(x_0) \leqslant f(\lambda x_1 + (1-\lambda)x_0) \leqslant \lambda f(x_1) + (1-\lambda) f(x_0) < f(x_0) \tag{2.21}$$

which is a contradiction. □

The set of points that minimize a convex function is always convex and closed if the set on which f is real-valued is closed, i. e., f is closed. (The convexity is immediate. Let x_0 be a minimum of f. The complement of the set of minima is equal to $\{x \mid f(x) > f(x_0)\}$ which is open, because f is continuous in the set $\{x \mid f(x) < \infty\}$.)

2.1.3. Differentiability

From now on, we restrict ourselves to finite-valued functions, i. e., to functions $f : X \to \mathbb{R}$.

Convex functions are always one-sided differentiable in every direction. Because this result is non-trivial, we formulate it as a lemma before using the existence of the differential in the next definition.

Lemma 1. *Let $f : X \to \mathbb{R}$ be convex, $x \in X^\circ$, where X° denotes the interior of X, and $v \in H$. Then the limit*

$$\lim_{h \searrow 0} \frac{f(x + hv) - f(x)}{h} \tag{2.22}$$

exists in $\mathbb{R}$.

Proof. Can be found in [HUL93] p. 238 (for the case $X = \mathbb{R}^n$, but the proof for the general case is the same). □

Definition 4. *Let $f : X \to \mathbb{R}$ be convex, $x \in X^\circ$ and $v \in H$. The real number*

$$D_v f(x) = \lim_{h \searrow 0} \frac{f(x + hv) - f(x)}{h} \tag{2.23}$$

is called the directional derivative of f at x in direction v. □

For a real convex function f, we essentially have only two directions: $v = +1$ and $v = -1$. The directional derivative $D_{+1}f(x)$ is equal to the right-sided derivative $f'_+(x)$ of f at x. Likewise, $D_{-1}f(x)$ is the negative of the left-sided derivative $-f'_-(x)$. For arbitrary $v > 0$, we have

$$\begin{aligned} D_v f(x) &= \lim_{h \searrow 0} \frac{f(x + hv) - f(x)}{h} && (2.24)\\ &= v \lim_{hv \searrow 0} \frac{f(x + hv) - f(x)}{hv} && (2.25)\\ &= v f'_+(x). && (2.26) \end{aligned}$$

Analogously, for $v < 0$:

$$D_v f(x) = v f'_-(x) \tag{2.27}$$

Of course, we always have ($v = 0$):

$$D_0 f(x) = 0 \tag{2.28}$$

A real function g is differentiable at x if and only if $g'_+(x)$, $g'_-(x)$ exist and are equal. Hence, if f is differentiable, then

$$D_v f(x) = v f'(x). \tag{2.29}$$

Definition 5. *Let $f : X \to \mathbb{R}$ be convex and $x \in X^\circ$. The set*

$$\partial f(x) = \{s \in H \mid \langle s, v \rangle \leqslant D_v(x) \text{ for all } v \in H\} \tag{2.30}$$

is called the subdifferential of f at x. □

We see immediately that for differentiable real functions f, we have $\partial f(x) = \{f'(x)\}$, because setting v to $+1$ resp. -1 in (2.29) and the definition of the subderivative gives us for $s \in \partial f(x)$

$$s \leqslant f'(x) \tag{2.31}$$

resp.

$$-s \leqslant -f'(x), \tag{2.32}$$

hence $s = f'(x)$. Moreover, $f'(x)$ is in $\partial f(x)$ because of (2.29). This is part of a more general principle:

Proposition 3. *Let $f : X \to \mathbb{R}$ be convex and $x \in X$. Then holds:*

(1) *$\partial f(x)$ is a nonempty convex compact set.*

(2) *$\partial f(x) = \{s_0\}$ for an $s_0 \in H$ if and only if f is differentiable in x. In this case, we have $\nabla f(x) = s_0$.*

Proof. [HUL93] p. 239. □

Theoretically very interesting, though of no great importance to our problem, is the following

Theorem 2. *Let $X \subset \mathbb{R}^n$ be open and $f : X \to \mathbb{R}$ be a convex function. Then f is differentiable Lebesgue almost everywhere.*

Proof. In fact, for $n = 1$, it is $\partial f(x) = [f'_-(x), f'_+(x)]$. Thus, if f is not differentiable at x, then $\partial f(x)$ is a nontrivial interval and hence contains a rational number in its interior. Further, for $x < y$, the interiors of $\partial f(x)$ and $\partial f(y)$ are disjoint. This shows that the set of points where f fails to be differentiable is at most countable and thus Lebesgue zero. For details, refer to [HUL93] pp. 189-190. □

With the notion of the subdifferential, we can formulate our first minimality criterion (which is still very abstract by now).

2.1.4. A Minimality Condition

Theorem 3. *Let $f : X \to \mathbb{R}$ be convex and $x_0 \in X$. The following are equivalent.*

(1) *x_0 minimizes f, i. e., $f(x_0) \leqslant f(x)$ for all $x \in X$*

(2) $0 \in \partial f(x_0)$

Proof. ($\Rightarrow$): It is $f(x_0 + hv) - f(x_0) \geqslant 0$ for all hv and thus $D_v f(x) \geqslant 0$ for all v.
($\Leftarrow$): We show that the mapping

$$h \mapsto \frac{f(x_0 + hv) - f(x_0)}{h} \tag{2.33}$$

is nondecreasing for $h > 0$. Let $0 < h_1 < h_2$. It is

$$\begin{aligned} f(x_0 + h_1 v) &= f\left((1 - \frac{h_1}{h_2})x_0 + \frac{h_1}{h_2}(x_0 + h_2 v)\right) && (2.34)\\ &\leqslant (1 - \frac{h_1}{h_2})f(x_0) + \frac{h_1}{h_2} f(x_0 + h_2 v) && (2.35)\\ &= f(x_0) + h_1 \frac{f(x_0 + h_2 v) - f(x_0)}{h_2} && (2.36) \end{aligned}$$

which yields

$$\frac{f(x_0 + h_1 v) - f(x_0)}{h_1} \leqslant \frac{f(x_0 + h_2 v) - f(x_0)}{h_2}. \tag{2.37}$$

This implies that

$$\lim_{h \searrow 0} \frac{f(x_0 + hv) - f(x_0)}{h} = \inf_{h>0} \frac{f(x_0 + hv) - f(x_0)}{h} \tag{2.38}$$

where the left-hand side is $\geqslant 0$ for all $v \in X$. This already shows the minimality of $f(x_0)$: We can set $v = x - x_0$, $h = 1$ and get $f(x) - f(x_0) \geqslant 0$ with equation (2.38). □

2.2. Bayesian Classification

In supervised learning, the situation is the following. A number of objects (commonly described as real-valued vector) together with a label (a real number or an element of $\{0, 1\}$) are presented to the machine. This set of pairs (object, label) is called the *training set* T. The machine's objective is to derive a *decision function* f that maps objects to labels such that it is consistent with the training set (i. e., f(object)=label for all (object, label) $\in T$) and that it predicts labels for new objects *well enough*. In fact, the requirement that f is consistent with *all* elements of T is often dropped and replaced by the requirement that it is *close enough* for *most* elements of T.
For the procedure of finding the decision function f, two cases can be distinguished ([Vap98]):

1. It is known that the "real" labeling function is taken from a fixed set $\Gamma = \{f_\alpha \mid \alpha \in A\}$ of functions.

2. No such set is known.

We will be concerned with the first case, i. e., we are given such a set Γ and we only adjust the parameter α to find the function that fits best. Both approaches assume that such a "real" labeling function exists. One can of course generalize this and search for an appropriate *probability distribution* for the labeling process.
We suppose that the labeling process by the *supervisor* that labeled the training set is determined by a fixed (but unknown) probability distribution F. Let $F(\omega|x)$ denote the probability that the supervisor assigns label ω to the object x and let $F(x)$ denote the probability that object x is chosen for classification. The problem of finding the best parameter α is then minimizing the function

$$R(\alpha) = \int L(f_\alpha(x), \omega) dF(x, \omega) \tag{2.39}$$

where $F(x, \omega) = F(x)F(\omega|x)$ and L is an appropriately chosen *loss function*, i. e., a non-negative function that increases with the mislabelings of the classification function f_α. A very simple loss function would be $L(a, b) = 1 - \delta_{a,b}$ where δ denotes the Kronecker symbol, i. e., $\delta_{a,b} = 1$ if and only if $a = b$ and $= 0$ else.
If we only take in account the only thing we know about the distribution F – namely the training set $T = \{(x_1, \omega_1), \ldots, (x_\ell, \omega_\ell)\}$ – then (2.39) becomes

$$R'(\alpha) = \frac{1}{\ell} \sum_{i=1}^{\ell} L(f_\alpha(x_i), \omega_i). \tag{2.40}$$

Here we assume that the training set was chosen with the same probability distribution F as the forthcoming samples.
Since what we do is *binary* classification, i. e., there are only two possible labels for all objects, we can have $\omega \in \{-1, 1\}$ and by using a simple loss function, we get

$$R''(\alpha) = \sum_{i=1}^{\ell} |f_\alpha(x_i) - \omega_i| \tag{2.41}$$

as the risk function we are to minimize. In fact, up to a constant factor and constant summand, the simple loss function $(1 - \delta)$ is the only one in binary classification if we demand L to be symmetric.

2.3. Constrained Quadratic Optimization for SVMs

In this and the following section we discuss methods for solving the general optimization problem from definition 1 and explore the problem we are to solve in Support Vector Training. Up to now, we neglected the feasibility set S, that is we assumed $S = \mathbb{R}^n$. As in ordinary vector analysis, finding extremal points in non-open sets is a lot harder than it is in open sets where we have a convenient necessary condition ($\nabla f = 0$). For the case that the feasibility set is a differentiable manifold there is the well-known method of *Lagrangian multipliers* ($\nabla f = \sum \lambda_i \nabla \varphi_i$) for which we will derive a passable substitution (in reality, even a generalization) in the next section. In our case, S will be a closed convex set and will be described by equality and inequality *constraints*. The feasibility set will be expressed as the intersection of inverse images of closed sets under convex functions.
Suppose our training set is

$$T = \{(x_1, \omega_1), \ldots, (x_\ell, \omega_\ell)\} \tag{2.42}$$

where $\omega_i \in \{-1, +1\}$ and $x_i \in \mathbb{R}^n$ for $1 \leqslant i \leqslant \ell$. We split T into the two sets

$$A = \{x_i \mid (x_i, \omega_i) \in T, \omega_i = +1\}, \tag{2.43}$$
$$B = \{x_i \mid (x_i, \omega_i) \in T, \omega_i = -1\}. \tag{2.44}$$

Our goal is to find an affine hyperplane V in $\mathbb{R}^n$ such that the points of A are on one side and the points of B are on the other. Of course, this is only possible if $A \cap B = \emptyset$, i. e., no point has *both* labels. Additionally, this hyperplane should have maximal distance to the points of $A \cup B$. More formally, if V is described by the equation

$$\langle x, w \rangle = c \tag{2.45}$$

where $w \in \mathbb{R}^n$, $\|w\| = 1$ and $c \in \mathbb{R}$, then it should hold that $\langle x, w \rangle > c$ for $x \in A$ and $\langle y, w \rangle < c$ for $y \in B$. In this case, we say that V *separates the sets A and B* and that A and B are *separable*. The distance $\operatorname{dist}(x, V)$ of a point x to V is equal to $|c - \langle x, w \rangle|$. More explicitly for our points of interest,

$$\operatorname{dist}(x, V) = \begin{cases} \langle x, w \rangle - c & , x \in A \\ c - \langle x, w \rangle & , x \in B \end{cases}. \tag{2.46}$$

The following exposition closely follows [Vap98], chap. 10 and [Bur98], chap. 3. We define for every normed w the numbers $c_1(w) = \inf\{\langle x, w \rangle \mid x \in A\}$ and $c_2(w) = \sup\{\langle y, w \rangle \mid y \in B\}$. According to (2.46), the sum of the minimal distances of A and B to V respectively is equal to $\rho(w) = (c_1(w) - c) + (c -$

$c_2(w)) = c_1(w) - c_2(w)$. We note that even though the parameter c is yet to be determined, the expression $\rho(w)$ does not depend on it. Since we want the distances to be maximal, we want to maximize $\rho(w)$. After we found such a normed w for which $\rho(w)$ is maximal, it is clear that $c = (c_1(w) + c_2(w))/2$ is the optimal choice for the parameter c. For it is the mean of both extreme values.

Lemma 2. *The function $w \mapsto \rho(w)$ has a unique maximum if $A, B \neq \emptyset$ and $A \cap B = \emptyset$.*

Proof. The existence of a maximum is clear since S^{n-1} is compact and ρ is continuous. To prove the uniqueness, we first show that ρ, as a function on the set of $x \in \mathbb{R}^n$ for which $\|x\| \leqslant 1$, attains its maximum at the set's boundary, i. e., S^{n-1}. For let $0 < \|x\| < 1$, then $\rho(x/\|x\|) = \rho(x)/\|x\| > \rho(x)$ with $\|x/\|x\|\| = 1$. Let now $x_1, x_2 \in S^{n-1}$ be two distinct maxima. Then $(x_1+x_2)/2$ is also a maximum with $\|(x_1 + x_2)/2\| < 1$. Contradiction. □

We can reformulate the problem to finding $w \in \mathbb{R}^n \setminus \{0\}$ and $b \in \mathbb{R}$ such that

$$\langle x, w\rangle - b \geqslant +1 \qquad (x \in A) \tag{2.47}$$
$$\langle y, w\rangle - b \leqslant -1 \qquad (y \in B) \tag{2.48}$$

where $\|w\|$ is minimal. The above is equivalent to

$$\omega_i \left(\langle x_i, w\rangle - b\right) - 1 \geqslant 0 \qquad (1 \leqslant i \leqslant \ell). \tag{2.49}$$

If we find an optimal w, equality holds in (2.49) for at least two i – one of each class. Then, the distance between the two hyperplanes defined by $\langle x, w\rangle - b = \pm 1$ is equal to $2/\|w\|$. These hyperplanes are parallel to V which is defined by $\langle x, w\rangle - b = 0$ and are called *support hyperplanes*. Thus, the above found value $2/\|w\|$ is the same as that of the function ρ for the respective argument. To sum up, our problem to solve is now the following.

$$\min \|w\|^2 \quad \text{subject to (2.49)} \tag{2.50}$$

This problem is a convex optimization problem since the objective function $x \mapsto \|x\|^2$ is convex and so is the feasibility set S which is the intersection of half spaces defined by (2.49).

2.4. Are we there yet? – Optimality Conditions

In this section, we will introduce necessary and sufficient conditions for minima of convex optimization problems. These are known as the *Karush-Kuhn-Tucker (KKT)* conditions.

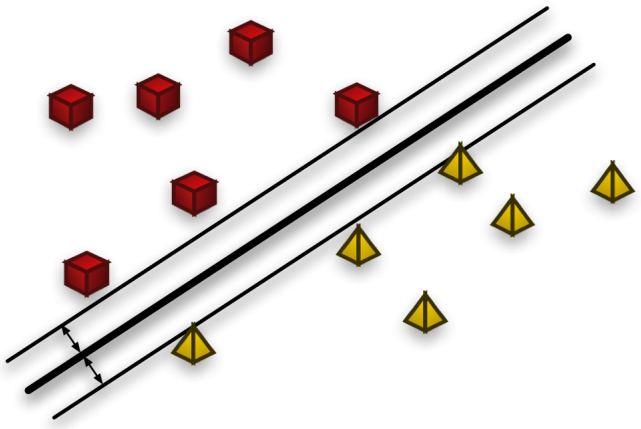

Figure 2.3.: Support hyperplanes

Theorem 4 (KKT)**.** *Let $f_i : X \to \mathbb{R}$ be convex functions for $0 \leqslant i \leqslant m$ where $X \subset \mathbb{R}^n$ is a convex set. We define*

$$S = \{x \in X \mid f_i(x) \leqslant 0 \quad (1 \leqslant i \leqslant m)\}. \tag{2.51}$$

Suppose that x^* minimizes f_0 in the set S. Then there exists $(\lambda_0^*, \lambda^*) = (\lambda_0^*, \ldots, \lambda_m^*) \neq 0$ such that

(1) *the function $L(x, \lambda_0^*, \lambda^*) = \sum_{i=0}^m \lambda_i^* f_i(x)$ is minimized by x^* in the set X.*

(2) *$(\lambda_0^*, \lambda^*) \geqslant 0$.*

(3) *$\lambda_i^* f_i(x^*) = 0$ for all i.*

Let now x^* and (λ_0^*, λ^*) satisfy conditions (1), (2), (3). If $\lambda_0^* \neq 0$, then x^* minimizes f_0 in S. □

The function

$$L(x, \lambda_0, \lambda) = \sum_{i=0}^{m} \lambda_i f_i(x) \tag{2.52}$$

is called the *Lagrangian* of the optimization problem.

Proposition 4. *With the notation of theorem 4, it is sufficient for λ_0^* to be possibly $\neq 0$ that there exists an $x_0 \in X$ with $f_i(x_0) < 0$ for all $1 \leqslant i \leqslant m$.* □

Thus, in this case, we can eliminate λ_0 from our Lagrangian by division and get the simpler version

$$L(x, \lambda) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x). \tag{2.53}$$

2.5. Kernel Functions

It is possible to replace the separating hyperplane by a more general hypersurface in $\mathbb{R}^n$ which is the inverse image of a linear hyperplane in a higher (often infinite) dimensional Hilbert space H. The technique we will use is known as the *kernel trick*. It enables us to classify data sets that are not separable by a (linear) hyperplane.

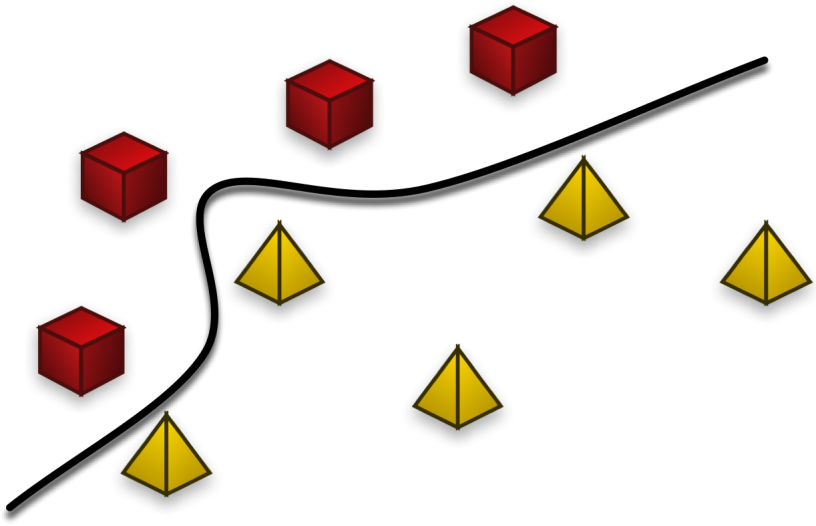

Figure 2.4.: Data that is not linearly separable

We consider a transformation $\Phi : \mathbb{R}^n \to H$ with which we map the training data before applying the algorithm. Thus, the inner products that occur become $\langle \Phi(x), \Phi(y) \rangle = K(x, y)$. We see that we do not really have to know the mapping Φ – not even the space H. We only need to know the *kernel function* K.

Now, one can go the opposite direction and ask: For which functions $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ exists a Hilbert space $(H, \langle \cdot, \cdot \rangle_H)$ and a transformation $\Phi : \mathbb{R}^n \to H$, such that $K(x, y) = \langle \Phi(x), \Phi(y) \rangle_H$?
This question is partially answered by *Mercer's theorem* (in [Mer09], although this version is a generalization of his original theorem to more general domains than the real compact intervals $[a, b]$). It states that this is true for all continuous symmetric K that are nonnegative definite. The following proof can be skipped.

Theorem 5 (Mercer)**.** *Let $(X, \mathfrak{A}, \mu)$ be a σ-finite measure space (i. e., for $\iota \in \mathbb{N}$ there exist $A_\iota \in \mathfrak{A}$ with $\mu(A_\iota) < \infty$ such that $X = \bigcup A_\iota$). Let further K :*

$X \times X \to \mathbb{R}$ be a symmetric function in $L^2(X^2)$ such that for all $f \in L^2(X)$ there holds:

$$\int_{X^2} K(x,y)f(x)f(y)d(x,y) \geqslant 0 \tag{2.54}$$

Then there exists an orthonormal family $(f_i)_{i\in I}$ in $L^2(X)$ and a family $(\lambda_i)_{i\in I}$ of non-negative real numbers such that

$$K(x,y) = \sum_{i\in I} \lambda_i f_i(x) f_i(y) \tag{2.55}$$

for almost all $(x,y) \in X^2$.

Proof. We define the operator $T = T_K : L^2(X) \to L^2(X)$ by

$$T_K f(x) = \int_X K(x,t)f(t)dt. \tag{2.56}$$

It maps into $L^2(X)$, because K is in $L^2(X^2)$. For $L^2(X)$ is a Hilbert space, there exists an orthonormal basis B of $L^2(X)$. We will show that $\sum \|Tb\|^2$ is finite if b varies in B:

$$\begin{aligned}
\sum_{b\in B} \|Tb\|^2 &= \sum_{b\in B} \int_X \left| \int_X K(x,t)b(t)dt \right|^2 dx && (2.57)\\
&= \int_X \sum_{b\in B} |\langle K(x,\cdot), b\rangle|^2 \, dx && (2.58)\\
&= \int_X \|K(x,\cdot)\|^2 \, dx && (2.59)\\
&= \int_X \int_X |K(x,y)|^2 \, dydx < \infty && (2.60)
\end{aligned}$$

This shows that T is Hilbert-Schmidt and hence compact ([Wei00], Satz 3.18(b)). Since K is symmetric, so is T. We can thus apply the spectral theorem and get the existence of an orthonormal basis $(f_i)_{i\in I}$ of $L^2(X)$ which consists of eigenfunctions of T. Let $Tf_i = \lambda_i f_i$ for $i \in I$. It is $\lambda_i = \lambda_i \langle f_i, f_i\rangle = \langle Tf_i, f_i\rangle \geqslant 0$ for all i. Further, for $f \in L^2(X)$, it is $\int f(t) \sum \lambda_i f_i(t) f_i(x) dt = \sum \lambda_i f_i(x) \int f_i(t) f(t) dt = \sum f_i(x) \langle T_K f_i, f\rangle = \sum \langle T_K f, f_i\rangle f_i(x) = T_K f(x)$ almost everywhere. Now, the mapping $K \mapsto T_K$ is injective since X is σ-finite. Hence, the claimed formula follows. □

We can then define the transformation $\Phi : X \to \ell^2(I)$ by

$$\Phi(x) = \left(\sqrt{\lambda_i} f_i(x)\right)_{i\in I} \tag{2.61}$$

where our Hilbert space $H = \ell^2(I)$ is the space of quadratic summable real sequences with index set I equipped with the inner product

$$\langle a, b\rangle = \sum_{i\in I} a_i b_i \tag{2.62}$$

where $a = (a_i)$, $b = (b_i)$. The image $\Phi(x)$ is really in $\ell^2(I)$, because

$$\sum_{i\in I} \left|\sqrt{\lambda_i} f_i(x)\right|^2 = \sum_{i\in I} \lambda_i f_i(x) f_i(x) = K(x,x). \tag{2.63}$$

In particular, $\Phi(x)$ is quadratic summable. This gives us our desired result

$$K(x,y) = \langle \Phi(x), \Phi(y)\rangle . \tag{2.64}$$

Our favored space $\mathbb{R}^n$ is σ-finite (with respect to the Lebesgue measure λ). We can choose $A_\iota = \{x \in \mathbb{R}^n \mid \|x\| < \iota\}$. It is $\lambda(A_\iota) = \iota^n \pi^{n/2}/\Gamma(\frac{n}{2}+1) < \infty$ and $A_\iota \to \mathbb{R}^n$.

Possible nonnegative definite kernels include ([Bur98])

- $K(x,y) = (\langle x, y\rangle + 1)^p \quad (p \in \mathbb{N})$
- $K(x,y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}} \quad (\sigma \neq 0)$
- $K(x,y) = \tanh(\kappa\langle x, y\rangle - \delta)$ for *certain* $\kappa, \delta \in \mathbb{R}$

where we might have to restrict K to a smaller set than the whole $\mathbb{R}^n$. For example, the *Gaussian* kernel $K(x,y) = e^{-\|x-y\|^2/2\sigma^2}$ is not in $L^2(\mathbb{R}^n \times \mathbb{R}^n)$:

$$\begin{aligned}
\int_{\mathbb{R}^n\times\mathbb{R}^n} |K(x,y)|^2 \, d(x,y) &= \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} e^{-\|x-y\|^2/\sigma^2} dx dy && (2.65)\\
&= \int_{\mathbb{R}^n}\int_{\mathbb{R}^n} e^{-\|x\|^2} |\sigma|^n \, dx dy && (2.66)\\
&= \int_{\mathbb{R}^n} \left|\sigma\sqrt{\pi}\right|^n dy = \infty && (2.67)
\end{aligned}$$

But it is, of course, in $L^2(C \times C)$ for every compact subset C of $\mathbb{R}^n$. ($\int K^2 \leqslant \lambda(C)^2 \max K^2$)

3. SVM Algorithms

This chapter introduces the usual formulation of the SVM training problem and reviews popular solution algorithms. Caching and shrinking techniques are also treated though they are not applicable for use on the microcontroller due to the stringent memory space limitations. In fact, one could skip all the preceding and start with section 3.3, the final optimization problem statement, if one is only interested in the algorithmic aspects of Support Vector Machines as opposed to the mathematical aspects and derivations.

3.1. Naïve KKT

With our convex constraint functions (2.49), $f_i(w,b) = -\omega_i(\langle x_i, w\rangle - b) + 1$ for $1 \leqslant i \leqslant \ell$, and a slightly modified objective function $f_0(w,b) = \frac{1}{2}\|w\|^2$, the simple Lagrangian (2.53) becomes

$$L(w,b,\lambda) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{\ell} \lambda_i \omega_i (\langle x_i, w\rangle - b) + \sum_{i=1}^{\ell} \lambda_i. \tag{3.1}$$

The KKT conditions tell us that it is necessary for (w,b) to be a solution of (2.50) that (w,b) is a minimum of the Lagrangian for a certain choice of λ. Since L is continuously differentiable, it is therefore necessary that the partial derivatives $\partial L/\partial w$ and $\partial L/\partial b$ vanish. This means that $w - \sum \lambda_i \omega_i x_i = 0$, which is

$$w = \sum_{i=1}^{\ell} \lambda_i \omega_i x_i. \tag{3.2}$$

Also, by partial derivation with respect to b,

$$\sum_{i=1}^{\ell} \lambda_i \omega_i = 0. \tag{3.3}$$

We can substitute this into the Lagrangian by noticing

$$\frac{1}{2}\|w\|^2 = \frac{1}{2}\left\langle \sum \lambda_i \omega_i x_i, \sum \lambda_i \omega_i x_i \right\rangle = \frac{1}{2}\sum_{i,j=1}^{\ell} \lambda_i \lambda_j \omega_i \omega_j \langle x_i, x_j\rangle \tag{3.4}$$

and

$$\sum_{i=1}^{\ell} \lambda_i \omega_i (\langle x_i, w \rangle + b) = \sum_{i,j=1}^{\ell} \lambda_i \lambda_j \omega_i \omega_j \langle x_i, x_j \rangle + 0 \tag{3.5}$$

which yields

$$W(\lambda) = L(w, b, \lambda) = \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \lambda_i \lambda_j \omega_i \omega_j \langle x_i, x_j \rangle . \tag{3.6}$$

This new formulation of the Lagrangian does not depend on w and b anymore. Since it is necessary that (w, b) minimizes L for a choice of λ subject to certain constraints and since we know that a minimum exists, it is sufficient to maximize W with respect to λ subject to the same constraints. This reformulation, the so-called *dual* formulation, is summarized in section 3.3.

3.2. Dealing with Errors

There may be cases where we want to tolerate some training errors, i. e., points of the training set that lie on the wrong side of the hyperplane. This is the case, for example, if we use a linear classifier and the subsets of the training set that correspond to the respective labels are not linearly separable (i. e., there exists no hyperplane that separates the two sets). To achieve this, we introduce a penalty parameter C for points that fail to be on the side of its label. Of course, we want the penalty to be higher the greater the distance of the erroneous points to the hyperplane. But first of all, we need to loosen the strict constraint $\omega_i(\langle x_i, w \rangle + b) \geqslant 1$. We therefore introduce non-negative slack variables ξ_i ([Bur98], 3.5) to transform the above into

$$\omega_i(\langle x_i, w \rangle - b) \geqslant 1 - \xi_i \qquad (1 \leqslant i \leqslant \ell) \tag{3.7}$$

where we want to have $\xi \geqslant 0$. To actually implement the penalty, we simply add the slack variables to the objective function of the minimization problem multiplied with the parameter C:

$$f_0(w, b, \xi) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{\ell} \xi_i \tag{3.8}$$

What changes does this introduce into the dual formulation? Well, the full Lagrangian reads as follows (note the changed/additional constraints in the primal formulation!).

$$L(w, b, \xi, \lambda, \mu) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{\ell} \xi_i - \sum_{i=1}^{\ell} \lambda_i (\omega_i (\langle x_i, w \rangle - b) - 1 + \xi_i) - \sum_{i=1}^{\ell} \mu_i \xi_i \tag{3.9}$$

Again, this can be simplified by setting $\partial L/\partial w = \partial L/\partial b = 0$. These equations yield the same as above. Additionally, we can have $\partial L/\partial \xi_i = 0$ for $1 \leqslant i \leqslant \ell$ which is $C - \lambda_i - \mu_i = 0$. This implies

$$C \sum_{i=1}^{\ell} \xi_i - \sum_{i=1}^{\ell} \lambda_i \xi_i - \sum_{i=1}^{\ell} \mu_i \xi_i = 0. \tag{3.10}$$

Thus, in reality ξ and μ do not appear at all in the dual Lagrangian $W(\lambda)$. The only additional constraint we get, since $\mu_i \geqslant 0$, is $C \geqslant \lambda_i$.

3.3. The Dual Formulation

We already utilize section 2.5 here, i. e., we replace the scalar product $\langle x, y \rangle$ by a kernel function $K(x, y)$. Further, we again state it as a minimization problem by flipping signs of the objective function. The complete dual formulation then reads:

$$\min \qquad W(\lambda) = -\sum_{i=1}^{\ell} \lambda_i + \frac{1}{2} \sum_{i,j=1}^{\ell} \lambda_i \lambda_j \omega_i \omega_j K(x_i, x_j) \tag{3.11}$$

$$\text{s. t.} \qquad \sum_{i=1}^{\ell} \lambda_i \omega_i = 0 \tag{3.12}$$

$$0 \leqslant \lambda \leqslant C \tag{3.13}$$

In the above formula, $0 \leqslant \lambda \leqslant C$ means $0 \leqslant \lambda_i \leqslant C$ for all i. With the notation $Q = (\omega_i \omega_j K(x_i, x_j))_{1 \leqslant i,j \leqslant \ell}$ and $e = (1)_{1 \leqslant i \leqslant \ell}$, it becomes

$$\min \qquad W(\lambda) = -\lambda^T e + \frac{1}{2} \lambda^T Q \lambda \tag{3.14}$$

$$\text{s. t.} \qquad \lambda^T \omega = 0 \tag{3.15}$$

$$0 \leqslant \lambda \leqslant Ce \tag{3.16}$$

where z^T denotes the transpose of z. The necessary and sufficient KKT conditions for a minimum are ([Pla98]):

$$\lambda_i = 0 \iff \omega_i u_i \geqslant 1 \tag{3.17}$$

$$0 < \lambda_i < C \iff \omega_i u_i = 1 \tag{3.18}$$

$$\lambda_i = C \iff \omega_i u_i \leqslant 1 \tag{3.19}$$

Here u_i denotes the "raw" classifier function evaluated at the training point x_i, that is

$$u_i = \sum_{j=1}^{\ell} \lambda_j \omega_j K(x_i, x_j) - b. \tag{3.20}$$

3.4. Solution Strategies

In this section, we briefly discuss popular algorithms for SVM training. Note that these are geared towards very large sets of input data and thus mostly not immediately utilizable for microcontroller use where we do not expect such magnitude of data. Also, these algorithms assume that almost an infinite amount of data can temporarily be stored on a hard disk drive and that only RAM space is limited, which is not true in an MCU environment.

3.4.1. Osuna et al.

This algorithm ([OFG97]) utilizes a *decomposition* of the input index set $\{1, \dots, \ell\}$ into the *working set* B and the remainder set N, whose associated multipliers λ_i will not change in the current iteration. If we denote by λ_J, ω_J and Q_{IJ} the vectors and matrices with entries corresponding to the index sets $I, J \subset \{1, \dots, \ell\}$, then the optimization problem becomes

$$\begin{aligned} \min \quad & -\lambda_B^T e + \frac{1}{2}\lambda_B^T Q_{BB}\lambda_B + \lambda_B^T q_{BN} && (3.21)\\ \text{w. r. t.} \quad & \lambda_B && (3.22)\\ \text{s. t.} \quad & \lambda_B^T \omega_B + \lambda_N^T \omega_N = 0 && (3.23)\\ & 0 \leqslant \lambda_B \leqslant Ce && (3.24) \end{aligned}$$

Here, we omitted the constant terms that only include λ_N and Q_{NN}. Further, $q_{BN} = \left(\omega_i \sum_{j \in N} \lambda_j \omega_j K(x_i, x_j)\right)_{i \in B}$. The algorithm is now based on the following two observations.

- If we move an arbitrary index i from B to N, the objective function (of the original problem) does not change and the solution is feasible. *(Build down)*
- If we move an index i from N to B that violates the KKT conditions and solve the subproblem for B, there is a strict improvement of the objective function. *(Build up)*

The algorithm is sketched in pseudo-code below in figure 3.1.

Despite its good reception and reported positive results, the algorithm has a theoretical disadvantage: Though it is guaranteed that the solution improves in each iteration, there is no proof that it actually converges to an optimal solution ([CHL00]).

```
Osuna(x, ω) {
    choose B ⊂ {1, ..., ℓ} arbitrarily;
    N := {1, ..., ℓ} \ B;
    for(;;)
    {
        solve subproblem for B;
        if(∃i ∈ N such that λ_i violates KKT)
        {
            choose j ∈ B arbitrarily;
            B := {i} ∪ B \ {j};
            N := {j} ∪ N \ {i};
        } else break;
    }
    return λ; }
```

Figure 3.1.: Osuna et. al. decomposition algorithm

3.4.2. SMO

Sequential Minimal Optimization (SMO, [Pla98]) essentially employs the idea of Osuna's decomposition algorithm with $|B| = 2$ and adds heuristics for the choice of the next working set pair. The main advantage of having only two multipliers in the working set at a time is that the optimal solution can be computed analytically here and the algorithm therefore does not have to rely on the usage of numeric quadratic program solvers. We will take more time to explain and derive this method since we will use this algorithm in the implementation (chapter 4).

We consider the two Lagrangian multipliers λ_1 and λ_2. (We assume $B = \{1, 2\}$ without loss of generality.) The constraint (3.16) is $0 \leqslant \lambda_1, \lambda_2 \leqslant C$ and (3.15) means $\omega_1\lambda_1 + \omega_2\lambda_2 = \omega_1\lambda_1' + \omega_2\lambda_2'$ where λ_i' denotes the old value of λ_i of the previous iteration. Following [Pla98], we first compute the optimal value for λ_2 and then calculate λ_1 from the constraints. We distinguish the cases $\omega_1 = \omega_2$ and $\omega_1 \neq \omega_2$. In the first case, we have $\lambda_1 + \lambda_2 = d$, in the second $\lambda_1 - \lambda_2 = d$ where d is a constant. Thus, the possible values for (λ_1, λ_2) lie all on a line segment depicted in figure 3.2.

The lower and upper limits for λ_2 thus are: $L = \max(0, \lambda_2' + \lambda_1' - C)$, $H = \min(C, \lambda_2' + \lambda_1')$ for $\omega_1 = \omega_2$ and $L = \max(0, \lambda_2' - \lambda_1')$, $H = \min(C, C + \lambda_2' - \lambda_1')$

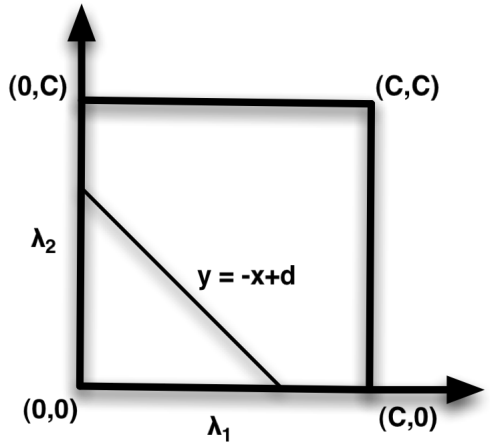

Figure 3.2.: Feasibility line for (λ_1, λ_2) in the case $\omega_1 = \omega_2$

for $\omega_1 \neq \omega_2$. The objective function (c. f. Osuna) is

$$-\lambda_1 - \lambda_2 + \frac{1}{2}K_{11}\lambda_1^2 + \frac{1}{2}K_{22}\lambda_2^2 + sK_{12}\lambda_1\lambda_2 + \lambda_1 v_1 + \lambda_2 v_2 \tag{3.25}$$

where $K_{ij} = K(x_i, x_j)$, $v_i = \sum_{j=3}^{\ell} \lambda_j' \omega_j K_{ij} = u_i + b' - \lambda_1' \omega_1 K_{1i} - \lambda_2' \omega_2 K_{2i}$ and $s = \omega_1\omega_2 = \pm 1$ depending on whether $\omega_1 = \omega_2$. By using $\lambda_1 + s\lambda_2 = \lambda_1' + s\lambda_2' = d$, we can transform this into a function that depends on λ_2 only. We can then set the ordinary first derivative equal to zero and calculate λ_2, provided $\eta = K_{11} + K_{22} - 2K_{12}$, which is the second derivative of this function, does not vanish. The embarrassing situation that it *does* vanish can occur, for example, if $x_i = x_j$ for $i \neq j$. In the other case, the optimal λ_2 is equal to

$$\lambda_2^{\text{new}} = \lambda_2' + \frac{\omega_2(u_1 - \omega_1 - u_2 + \omega_2)}{\eta}. \tag{3.26}$$

The quantity $E_i = u_i - \omega_i$ is called the error of the ith training example. Next we need to check whether $\lambda_2^{\text{new}} \in [L, H]$ and clip it into our square if it lies outside:

$$\lambda_2^{\text{new,clipped}} = \begin{cases} H & , \lambda_2^{\text{new}} > H \\ \lambda_2^{\text{new}} & , L \leqslant \lambda_2^{\text{new}} \leqslant H \\ L & , \lambda_2^{\text{new}} < L \end{cases} \tag{3.27}$$

We can then compute λ_1 from our equality constraint:

$$\lambda_1 = \lambda_1' + s\left(\lambda_2' - \lambda_2^{\text{new,clipped}}\right) \tag{3.28}$$

If $\eta = 0$, we just evaluate the objective function at the boundaries $\lambda_2 = L$ and $\lambda_2 = H$ and check whether the values differ. If so, we take the lower value. If not, then $\lambda_2 = \lambda_2'$ and we cannot make any progress here.

Two heuristics are used to determine the working set pair for the next iteration. The first heuristic is concerned with finding a suitable λ_2 and utilizes the fact that many multipliers end up being either 0 or C by termination of the algorithm. Thus, after a first pass through all examples, the algorithm consequently only chooses training examples where the corresponding multiplier is *strictly between* 0 and C. When there are no more changes possible with these examples, it returns to looping over *all* examples again. The second heuristic chooses a suitable partner for λ_2, i. e., one with the largest expected step size. To approximate the expected step size, the training example errors E_i are used. These are stored in an *error cache* for fast access. The choice with largest expected progress is the one where $|E_1 - E_2|$ is maximal. If there is no progress with this example, the algorithm first loops over all examples currently not at the bounds and then over all examples until progress is made.

3.4.3. SVM$^{\text{light}}$

T. Joachims introduced two new methods for solving the SVM training problem in [Joa98], where he presented the SVM$^{\text{light}}$ package. The first one is a more sophisticated method for selecting the working set than the "random" one used in Osuna decomposition. He proposed using first-order approximation to the objective function to find a direction d of steepest descent, in which the algorithm continues its operation. For this, he solves the following problem:

$$\begin{aligned}
\min \quad & V(d) = (\nabla W(\lambda))^T d && (3.29)\\
\text{s. t.} \quad & \omega^T d = 0 && (3.30)\\
& d_i \geqslant 0 \qquad (\lambda_i = 0) && (3.31)\\
& d_i \leqslant 0 \qquad (\lambda_i = C) && (3.32)\\
& -e \leqslant d \leqslant e && (3.33)\\
& |\{i \mid d_i \neq 0\}| = q && (3.34)
\end{aligned}$$

Here $q = |B|$ is the size of the working set. Our new working set will then be $\{i \mid d_i \neq 0\}$. Joachims gave an easy way to compute the solution of this optimization problem by sorting the λ_i in a clever way. The second method is "shrinking" – a technique that, like SMO's heuristic, uses the fact that there are many multipliers at the bounds in the optimal solution to reduce the size of the optimization problem. Of course, if the guess that a multiplier will be at the bounds was wrong, then it has to be visited in a later iteration nonetheless.

3.5. The Classifier Function

When we are done with the training algorithm and found our optimal λ and b, we are bound to ask how we can use this knowledge in classification of new examples. In the case of a *linear* SVM, i. e., no kernel function but the ordinary scalar product was used, we can simply calculate the vector w by

$$w = \sum_{i=1}^{\ell} \lambda_i \omega_i x_i \tag{3.35}$$

and the classification function is

$$f(x) = \operatorname{sgn}\left(\langle x, w\rangle - b\right). \tag{3.36}$$

The situation is not that easy if a kernel function was used. We cannot calculate w, because we do not even know the space H it belongs to. We therefore have to expand w in (3.36) to get

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{N_s} \lambda_i \omega_i K(x, x_i) - b\right) \tag{3.37}$$

where N_s denotes the number of *support vectors*, i. e., the number of vectors for which $\lambda_i \neq 0$. We assume here without loss of generality that the x_i are numbered in such a way that the first N_s vectors are the support vectors. These are the only ones we need to remember after the training process.

4. Implementation

This chapter describes the implementation of μSVM, a Support Vector Machine package for use on small microcontroller units.

4.1. μSVM Overview

Sequential Minimal Optimization (section 3.4.2) is utilized by the μSVM package for solving the quadratic program in SVM training. It supports the data types `char`, `int` and `float` for training example vectors. Support for new data types can easily be implemented with minimal changes. The decision for a specific type is made at compile time by macro definitions (e. g., compiler flag `-DuSVM_X_FLOAT` for `float` vectors, see Documentation for details). Different and also user-added kernel functions can be used for the algorithm. It can be changed at run-time by setting the function pointer `uSVM_ker`. Such a change can be useful, for example, if the timing requirements of the program change at run-time and the training procedure has to terminate earlier than in normal operation. Then, a faster kernel could be used. Also, the precision `uSVM_EPS` can always be changed. Since the final values of the Lagrangian multipliers are approximated fairly good early in the course of the training algorithm (see chapter 5), the value of the flag `uSVM_terminate` is checked periodically. The current multiplier values are output immediately and the algorithm is stopped in the case the flag was set. The main data structures are:

- `uSVM_x`: Pointer to the training example vectors. The dimension of the vectors is given by `uSVM_n` and the number of examples is stored in `uSVM_ell`. This field is accessed by the `uSVM_READ(i,k)` and `uSVM_WRITE(i,k,z)` macros.
- `uSVM_omega`: This array contains the labels of the example vectors. The values here should only be ± 1. The length of the array is again `uSVM_ell`.
- `uSVM_lambda`: Array of Lagrangian multipliers with `float` precision.
- `E`: The error cache described together with the SMO algorithm. It is used to determine the next working set pair. It also speeds up the train-

ing process by reducing the number of necessary kernel evaluations dramatically. Since this array consists of `uSVM_ell` floating point values, the memory requirements of the training process nearly doubles when using the error cache. It can therefore be deactivated by setting the `uSVM_NO_ERROR_CACHE` macro.

The relevant functions and procedures are:

- `uSVM_train()`: Starts the training algorithm. Allocates the `uSVM_lambda` field which is *not* `free()`'d by the function itself, but has to be freed by the application in case it is no longer used. Returns `-1` in case of an error and the number of support vectors otherwise. This number is also stored in the `uSVM_nSV` variable.
- `examine(i2)`: Searches for a suitable working set pair partner for `i2` until either progress is made or all examples were tried.
- `take_step(i1,i2)`: Computes the optimal solution for the subproblem induced by the indices `i1` and `i2`. Returns `1` if the current solution was improved and `0` otherwise.
- `uSVM_classify(x)`: Classifies the example given by the array `x` of dimension `uSVM_n`. Returns ± 1.

The training algorithm is sketched in figure 4.1 on page 29.

4.2. Target Hardware

μSVM was developed and tested on an Atmel AVR ATmega16 microcontroller using avr-libc version 1.2.5. The ATmega16 has a 16 MHz pipelined RISC processor with 1 kB internal RAM. The register size is 8 bit.

```
uSVM_train() {
    uSVM_lambda = malloc();
    while progress was made in previous iteration
        for all indices i2
            examine(i2);
    delete non-support vectors from uSVM_x, uSVM_omega and uSVM_lambda;
    nSV = # of support vectors;
    return nSV;
}

examine(i2) {
    while not tried all examples i1
        i1 = good working set partner for i2;
        if (take_step(i1,i2)==1) return 1;
    return 0;
}

take_step(i1,i2) {
    (lambda1,lambda2) = optimal solution for i1 and i2;
    if(lambda1==uSVM_lambda[i1] && lambda2==uSVM_lambda[i2])
        return 0;     // no progress
    update threshold b;
    update error cache E;
    uSVM_lambda[i1] = lambda1;
    uSVM_lambda[i2] = lambda2;
    return 1;
}
```

Figure 4.1.: μSVM training algorithm

5. Results and Discussion

This chapter evaluates the temporal behavior and the numerical accuracy of the μSVM package. Also, it is examined at which point in the runtime of the algorithm, the immediate results are sufficiently near to the final results to be useful. This is done because of the possibility to set the `uSVM_terminate` flag during the execution to force untimely termination.

5.1. Performance

We start this section by comparing the runtime of μSVM and Joachim's SVM$^{\text{light}}$ package on a personal computer for some example training sets. We also state the runtimes of μSVM on the ATmega16 microcontroller (section 4.2). We used four classes of example sets: EX A with $n = 5$ and $\ell = 5$, EX B with $n = 20$ and $\ell = 20$, EX C with $n = 50$ and $\ell = 10$. For each of these classes, three randomly chosen example sets were tested. For all these tests, we took a linear kernel with error penalty $C = 3.0$ and precision $\varepsilon = 0.005$. Additionally, the EX D.1 example set was chosen with the parameters $n = 10$ and $\ell = 30$. The PC tests were performed on a PowerMac G5 with two 2 GHz processors and 2.5 GB RAM. The tests on the MCU were performed one time using an error cache (with EC) and one time without (w/o EC). The package SVM$^{\text{light}}$ was not evaluated on the MCU because of the (PC-oriented) big memory consumptions which prohibit execution on the target hardware.

The results in table 5.1 on page 31 can lead to the following conclusions:

- μSVM performs quite well for small ℓ, even for big n, but greatly loses performance with the growth of ℓ.
- Growth of n affects operation with error cache much less than without error cache. This is because kernel evaluations are more expensive with big n.
- SVM$^{\text{light}}$ is more time efficient than μSVM on PC.

Next, we take a look at the numerical accuracy of μSVM. Therefore the results of μSVM on PC and SVM$^{\text{light}}$ are compared on the training example

Example set	SVM$^{\text{light}}$	μSVM on PC	μSVM with EC	μSVM w/o EC
EX A.1	0.010s	0.011s	2.7s	3.6s
EX A.2	0.009s	0.009s	2.4s	2.4s
EX A.3	0.010s	0.009s	3.6s	3.2s
EX B.1	0.017s	0.024s	88.2s	234.4s
EX B.2	0.019s	0.029s	170.0s	709.3s
EX B.3	0.018s	0.027s	196.0s	693.2s
EX C.1	0.011s	0.011s	10.8s	93.0s
EX C.2	0.012s	0.011s	13.1s	71.9s
EX C.3	0.012s	0.010s	9.9s	78.5s
EX D.1	0.121s	1.611s	$>$ 40.0min	–

Table 5.1.: Runtime of SVM implementations

sets and summarized in table 5.2. Stated is the *norm* of the error vector $v = (\lambda - \lambda^*, b - b^*)$ where (λ, b) is μSVM's solution and (λ^*, b^*) that of SVM$^{\text{light}}$. Also specified is the relative error $r = \|v\| \,/\, \|(\lambda^*, b^*)\|$. SVM$^{\text{light}}$ was chosen as the numerical reference implementation because of its excellent reputation in the community.

Example set	absolute error	relative error
EX A.1	0.00023377	0.0481%
EX A.2	0.00004444	0.0040%
EX A.3	0.00008987	0.0102%
EX B.1	0.00136866	0.2153%
EX B.2	0.00069989	0.2498%
EX B.3	0.00147428	0.4782%
EX C.1	0.00058508	0.3161%
EX C.2	0.00668395	1.8747%
EX C.3	0.00255609	1.6248%
EX D.1	0.00285187	0.0273%

Table 5.2.: Numerical errors of μSVM

5.2. Speed of Convergence

We investigate how fast the solutions converge to the final result. We therefore measured the relative error of the intermediate results of the algorithm with respect to the final values. Some results are depicted in figures 5.1 and 5.2. The

other examples draw a similar picture. We see that after 20% of the execution time, the error is less than 10%.

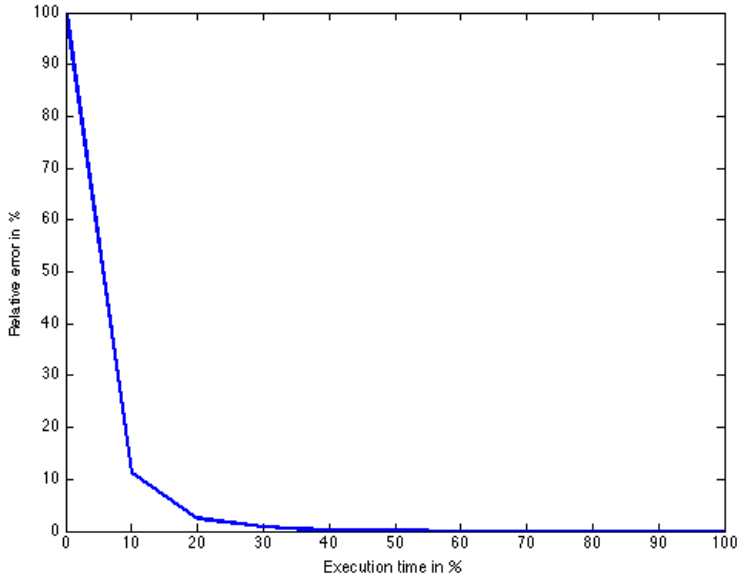

Figure 5.1.: Error progression of EX B.2

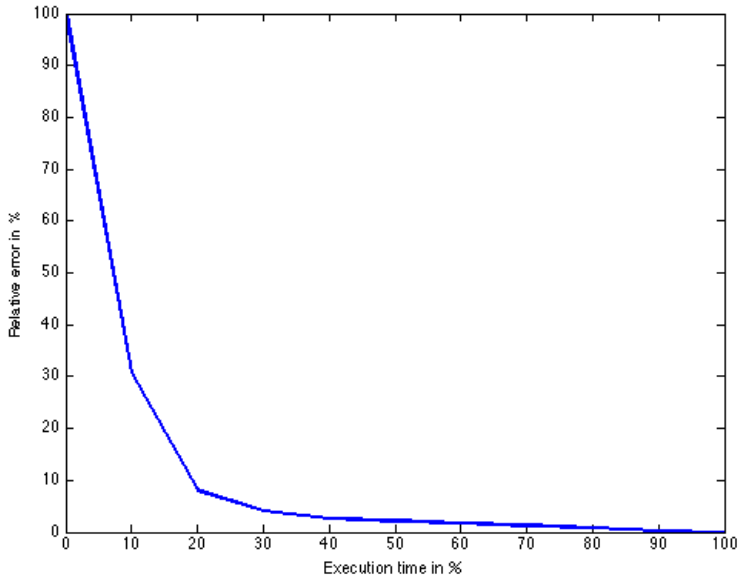

Figure 5.2.: Error progression of EX D.1

6. Summary

We started with a thorough introduction to convex analysis and derived the training process for Support Vector Machines. The Karush-Kuhn-Tucker conditions for solving constrained optimization problems, which are of great importance in practice, i. e., in the implementation, were explicitly written out in their general form and it was explained how these conditions can be applied to the SVM case.

An important part was generalizing the scalar product in $\mathbb{R}^n$ to kernel functions, i. e., functions that are scalar products in some other Hilbert space. Mecer's theorem, a sufficient condition for a function to be a kernel function, was stated and proved in a very general setting (σ-finite measure spaces), which is probably a novelty in SVM literature.

We then introduced the SVM implementations of Osuna at al., Sequential Minimal Optimization (SMO) and SVM$^{\text{light}}$. The SMO method was derived and investigated in more detail.

After illustrating the concepts of SVM training, we applied them in the implementation of μSVM. We have shown that despite the limited processing power and stringent memory space limitations in small microcontroller units, it is possible to use a fully-fledged SVM there. One problem is the long execution time of the implementation in the case of a large number of training examples. The experiments indicate, however, that it is often possible to stop the training process prematurely and still retain good numerical accuracy.

Future projects could focus on using μSVM in real-life applications or optimizing μSVM on other microcontroller units. Especially ones with more available memory space.

Bibliography

[Bur98] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[CHL00] C.-C. Chang, C.-W. Hsu, and C.-J. Lin. The analysis of decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 11(4):1003, July 2000.

[HUL93] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms I.* Springer, 1993.

[Joa98] T. Joachims. Making large-scale support vector machine learning practical. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Learning*. MIT Press, Cambridge, MA, 1998.

[Mer09] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, 209:415–446, 1909.

[OFG97] E. Osuna, R. Freund, and F. Girosi. Improved training algorithm for support vector machines. *Proceedings of the 1997 IEEE Workshop on Neural Networks for Signal Processing [1997] VII.*, pages 276–285, 24-26 Sep 1997.

[Pla98] J. Platt. Fast training of support vector machines using sequential minimal optimization. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Learning*. MIT Press, Cambridge, MA, 1998.

[Vap98] V. N. Vapnik. *Statistical Learning Theory.* Wiley, 1998.

[Wei00] J. Weidmann. *Lineare Operatoren in Hilberträumen 1.* Teubner, 2000.

A. Notation

$\mathbb{N}$	... set of positive integers
$\mathbb{R}$	... set of real numbers
$\overline{\mathbb{R}}$	... $\mathbb{R} \cup \{+\infty\}$
$B_r(x)$	... open ball with radius r and center x, i. e., $\{x \mid \|x\| < r\}$
S^{n-1}	... unit sphere in $\mathbb{R}^n$
inf	... infimum
sup	... supremum
$\lim a_i$	... limit of the net $(a_i)_i$
$\lim_{x \to x_0} f(x)$	... limit of the function f at x_0
$\lim_{x \searrow x_0} f(x)$	... right-sided limit of the function f at x_0
$\lim_{x \nearrow x_0} f(x)$	... left-sided limit of the function f at x_0
$\nabla f(x)$	... gradient of f at x
$f'(x)$	... derivative of f at x
$f'_+(x)$	... right-sided derivative of f at x
$f'_-(x)$	... left-sided derivative of f at x
$D_v f(x)$	... directional derivative of f at x in direction v
$\partial f(x)$	... subdifferential of f at x
$L^2(X)$	... space of quadratic integrable real-valued functions on X
$\ell^2(I)$	... space of quadratic summable real nets on I
$\langle x, y \rangle$	... inner product of x and y
$\|x\|$	... absolute value of x
$\|x\|$	... norm of x
x^T	... transpose of x
e	... $\sum 1/k!$
π	... $\Gamma(1/2)^2$
λ	... Lebesgue measure on $\mathbb{R}^n$
Γ	... gamma function, $x \mapsto \int_0^\infty t^{x-1} e^{-t} dt$
sgn	... signum function
tanh	... hyperbolic tangent function

B. μSVM Documentation

μSVM is a Support Vector Machine (SVM) implementation for use on microcontroller units (MCUs). The two main functions are:

- `int uSVM_train()`
- `char uSVM_classify(float *x)`

Their usage is described in the following sections.

B.1. Training

The training process is organized as follows.

1. Write the dimension n of the example vectors into `uSVM_n`.
2. Write the number ℓ of example vectors into `uSVM_ell`.
3. Allocate memory for $n \cdot (\ell + 1)$ vector components (either `char`, `int` or `float`, see B.3) and store the pointer in `uSVM_x`.
4. Allocate memory for ℓ `char` variables and store the pointer in `uSVM_omega`.
5. Write example vectors with the `uSVM_WRITE(i,k,z)` macro. Here, the `k`th component of the `i`th vector is written to `z`.
6. Write labels of example vectors into the `uSVM_omega` array. The values here can only be `+1` and `-1`.
7. Select the kernel function by setting the `uSVM_ker` function pointer to one of the functions described in B.1.1.
8. Select the precision `uSVM_EPS` and the penalty paramter `uSVM_C`.
9. Call `uSVM_train()`.

The return value of `uSVM_train()` is either `-1` in case of an error and the number of support vectors otherwise. This number is also stored in the `uSVM_nSV` variable.

B.1.1. Kernels

Currently available kernels are:

- `uSVM_scalar`: Linear kernel.
- `uSVM_gauss`: Gaussian kernel. The parameter σ can be modified with the `uSVM_GAUSS_SIGMA` macro. Default value is $\sigma = 1$
- `uSVM_poly`: Polynomial kernel. The exponent p can be modified with the `uSVM_POLY_P` macro. Default value is $p = 3$.

B.1.2. Early Termination

It is possible to terminate the training process ahead of time by setting the flag `uSVM_terminate`. This feature was added because experiments show that the computed values change only marginally after about 20% of the execution time.

B.2. Classification

After training, new vectors can be classified by calling `uSVM_classify(x)`, where `x` is an array of n `float` values. The return value is either `+1` or `-1` reflecting the label that is given to the vector by the SVM.

B.3. Compiler Flags

- `uSVM_X_INT` and `uSVM_X_FLOAT`: These flags select the data type used for training example vectors. Default is `char`.
- `uSVM_NO_ERROR_CACHE`: Disables the use of the error cache. The training algorithm needs less memory space, but is slower with this flag.
- `uSVM_GAUSS_SIGMA`, `uSVM_POLY_P`.

B.4. Example

```
uSVM_ker = uSVM_scalar;
uSVM_n = 5;
```

```
uSVM_ell = 5;
uSVM_C = 3.0;
uSVM_EPS = 0.005;
uSVM_x = malloc((uSVM_ell+1)*uSVM_n * sizeof(char));
uSVM_omega = malloc(uSVM_ell * sizeof(char));

uSVM_omega[0] = +1;
uSVM_x[0] = 2;
uSVM_x[1] = -3;
uSVM_x[2] = -1;
uSVM_x[3] = -5;
uSVM_x[4] = 2;
uSVM_omega[1] = -1;
uSVM_x[5] = 2;
uSVM_x[6] = -3;
uSVM_x[7] = 4;
uSVM_x[8] = -2;
uSVM_x[9] = -3;
uSVM_omega[2] = +1;
uSVM_x[10] = -1;
uSVM_x[11] = 0;
uSVM_x[12] = -1;
uSVM_x[13] = 4;
uSVM_x[14] = -5;
uSVM_omega[3] = +1;
uSVM_x[15] = -1;
uSVM_x[16] = 0;
uSVM_x[17] = -1;
uSVM_x[18] = 4;
uSVM_x[19] = -5;
uSVM_omega[4] = -1;
uSVM_x[20] = 1;
uSVM_x[21] = -1;
uSVM_x[22] = -1;
uSVM_x[23] = 4;
uSVM_x[24] = -1;

uSVM_train();

float *z = malloc(uSVM_n * sizeof(float));
z[0]=-1;
z[1]=0;
```

```
z[2]=-1;
z[3]=4;
z[4]=-5;

uSVM_classify(z);
free(z);
```